Санкт-Петербургский государственный университет

Кафедра системного программирования

Батов Никита Станиславович

Реализация кроссплатформенного доступа веб-приложения к системным интерфейсам

Бакалаврская работа

Научный руководитель:
ст. преп. Кириленко Я.А.

Рецензент:
Вице-президент по разработке программного обеспечения, Woodenshark LLC
Зиновьев Сергей Валерьевич

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Department of Software Engineering

Nikita Batov

Implementation of cross-platform web-application access to system API

Graduation Thesis

Scientific supervisor:
Senior Lecturer Kirilenko I.A.

Reviewer:
VP Software Engineering, Woodenshark LLC
Zinovev Sergei Valerevich

Saint-Petersburg
2016

Оглавление

Введение **4**

1. Постановка задачи **6**

2. Обзор предметной области **7**

2.1. Способы взаимодействия веб-приложения с системными интерфейсами 7

2.1.1. Универсальные технологии 7

2.1.2. Google Chrome и Chromium 8

2.1.3. Mozilla Firefox 8

2.1.4. Microsoft IE 9

2.1.5. Вывод 9

2.2. Системы автоматической трансляции 9

2.2.1. CrossBridge 9

2.2.2. Mandreel 10

2.2.3. Emscripten 10

2.2.4. Cheerp 10

2.2.5. Вывод 11

2.3. Использование libusb прикладными Linux-приложениями 12

2.4. Существующие решения 13

3. Описание решения **14**

3.1. Портирование libusb для работы в браузере 14

3.2. Внедрение libusb в Emscripten 18

3.3. Демонстрационное приложение на базе mspdebug 19

Заключение **20**

Список литературы **21**

Введение

По мнению аналитического агентства Gartner [7], к 2020 году интернет вещей объединит 26 миллиардов устройств, а доход поставщиков продуктов и услуг рынка интернета вещей составит к этому времени 300 миллиардов долларов. При таком росте отрасли требуется все больше квалифицированных специалистов [4] для разработки решений, объединяющих программные комплексы высокого уровня и программное обеспечение встраиваемых систем.

Одна из основных сложностей с привлечением молодых специалистов в эту область - это высокий порог входа. Даже для самых элементарных экспериментов требуется как аппаратная, так и программная части. Если в последние годы аппаратные решения, позволяющими создавать достаточно сложные робототехнические системы или решения для интернета вещей без специализированных знаний и умений, активно развивались и получили достаточно широкое распространение. То в области программного обеспечения, для работы со встраиваемыми системами, принципиальных изменений нет. Это по прежнему отдельные прикладные приложения, требующие установки на компьютер. Что не всегда возможно, в силу ограничений прав пользователя, это часто встречается в образовательных учреждениях, либо ограничений операционных систем таких как Chrome OS [19].

С современным развитием веб-технологий становится возможным создавать удобные и простые веб-приложения для разработки программного обеспечения для встраиваемых систем, доступ к котором осуществляется с помощью браузера. Но задача доставки программного обеспечения до аппаратной платформы остается актуальной, так как веб-приложения не могут напрямую взаимодействовать с системными интерфейсами (такими как USB и последовательный порт), так как это запрещено политиками безопасности практически всех браузеров. Пользователю требуется скачать на компьютер прошивку и затем используя специализированное программное обеспечение загрузить ее на аппаратную платформу. При такой системе все преимущества веб-приложения

не актуальны, так как все равно требуется установка каких-либо специальных приложений-загрузчиков. Таким образом возникает задача в реализации возможности доступа веб-приложения к системному интерфейсу USB, для автоматической загрузки прошивки на аппаратную платформу.

1. Постановка задачи

Целью данной работы является реализация возможности автоматической трансляции прикладных Linux-приложений, использующих только системные вызовы USB, в веб-приложения. Для достижения поставленной цели выделены следующие задачи:

- портировать библиотеку libusb для работы в браузере;
- внедрить портированную версию libusb в одну из систем автоматической трансляции;
- разработать демонстрационное веб-приложение на базе mspdebug.

2. Обзор предметной области

2.1. Способы взаимодействия веб-приложения с системными интерфейсами

За время развития современных веб-технологий многие компании пытались разработать метод доступа из веб-страницы к внутренним ресурсам клиентского компьютера. Цели подобных разработок были различны: от выполнения части кода как отдельного процесса операционной системы так и до прямого доступа к оборудованию.

Но основная проблема подобных решений заключается в безопасности: пользователь заходящий на веб-страницу не подозревает что его компьютер потенциально может быть атакован.

Так как одной из задач данной работы является портирование библиотеки libusb [23] для работы в браузере, то необходимо было провести исследование методов, предоставления доступа веб-приложений к системным интерфейсам.

2.1.1. Универсальные технологии

NPAPI [21] - программный интерфейс подключаемых модулей Netscape. Популярная в прошлом технология разработки переносимых кроссплатформенных модулей для браузеров. В настоящий момент не используется и отключена практически во всех популярных браузерах.

HTML5 [16] - новый стандарт разметки веб-страниц, был завершен в 2014 году. В контексте данной работы актуально нововведение в стандарте, касающее доступа к интерфейсу USB. Веб-приложение может получить доступ к ограниченному списку HID устройств: таких как игровой контроллер, микрофон или камера. Но использовать данную технологию для портирование библиотеки libusb невозможно, так как предоставляемый интерфейс является высокоуровневым, а использование низкоуровневых интерфейсов запрещено по соображениям безопасности.

2.1.2. Google Chrome и Chromium

Native Client [10] - технология, разрабатываемая Google, предназначенная для запуска машинного кода на клиентском компьютере. Другими словами, часть веб-приложения разрабатывается на C или C++ и исполняется напрямую без предварительных трансляций в JavaScript. Использование данной технологии в работе невозможно, так доступ к низкоуровневым интерфейсам отсекается на уровне анализа кода.

Native Messaging [11] - технология, позволяющая веб-приложению по определённому интерфейсу взаимодействовать с прикладным программным приложением, установленным на компьютере. Не актуальна в контексте данной работы, так требуется предварительная установка приложения на компьютер.

Chrome API [8] - технология, позволяющая веб-приложения получить доступ к низкоуровневым интерфейсам (таким как USB и последовательный порт) с разрешения пользователя. Причем предоставляемый интерфейс является достаточно низкоуровнемым. Другими словами разработчик получает непосредственный доступ ко всем потокам данным USB: bulk transfer, control transfer, interrupt transfer, isochronous transfer. Благодаря этому использование этой технологии делает возможным портирование libusb для работы в браузере.

WebUSB [17] - новая инициатива разработчиков Google, позволяющая любому веб-приложению получать доступ к системным интерфейсам из любого браузера. Была выпущена в апреле 2016 года и находится на стадии концепта. Данная разработка также подтверждает актуальность моей работы.

2.1.3. Mozilla Firefox

WebUSB [6] - технология, аналогичная Chrome API USB, находящая в разработке у Mozilla Foundation. На момент написания работы не была завершена.

2.1.4. Microsoft IE

ActiveX [18] - технология компания Microsoft для запуска машинного кода на клиентском компьютере. Дискредитировала себя в связи с проблемами с безопасностью.

2.1.5. Вывод

В результате исследования в качестве целевой технологии для обеспечения доступа к низкоуровневым интерфейсам была выбрана Chrome App API, как наиболее завершенная и документированная. Недостатком данного решения является ограничение на используемый браузер, а так же необходимое условие того, что веб-приложение должно быть размещено в Chrome App Store. В дальнейшем, с развитием инициативы WebUSB от разработчиков Google, планируется переход на данную технологию, как обеспечивающую доступ к низкоуровневым интерфейсам с любой веб-страницы.

2.2. Системы автоматической трансляции

Целью данной работы является реализация автоматической трансляции прикладных приложений в веб-приложения, поэтому в данной главе будет проведено сравнение систем позволяющих проводить source-to-source трансляцию исходного кода приложения на C/C++ в код на JavaScript. Для сравнения были выбраны следующие системы: Mandreel, Emscripten, Cheerp (бывший Duetto), CrossBridge.

2.2.1. CrossBridge

CrossBridge [1] - технология, разрабатываемая компанией Adobe, позволяющая транслировать приложения, написанные на C или C++ во Flash-приложения. Данная технология в последние года активно замещается. Поэтому использование ее в новых разработках не оправдано.

2.2.2. Mandreel

Mandreel - технология, ориентированная на портирование игр в браузер. Распространяется под платной проприетарной лицензией. Данная особенность накладывает финансовые ограничения на использование разрабатываемой технологии, что может отрицательно сказаться на ее дальнейшем развитии. Так же стоит учитывать, что использование технологии с закрытым исходным кодом нежелательно, так как это может привести к закрытию всех проектов, связанных с данной технологией, в случаи ее закрытия.

2.2.3. Emscripten

Emscripten [5] - транслятор, преобразующий байт-код LLVM, полученный при компиляции проекта на C/C++ или других языках, в код JavaScript или asm.js, выполняющийся в браузере. Используется Clang для сборки проекта в байт-код LLVM, и собственно инструментарий Emscripten для компиляции байт-кода в код JavaScript. Одной из особенностью, актуальной в контексте исследования, это полноценная поддержка синтаксиса C/C++. Другой важной деталью является модель памяти. В веб-приложение, полученном в результате трансляции, память представляется как один типизированный массив, в котором индексы являются абсолютными адресами. Данное решение негативно сказывается на производительности веб-приложения, так как работа с большими массивами в JavaScript происходит медленно. Но при этом использование данного подхода положительно сказывается на переносимости кода, так как данная модель является наиболее близкой к нативной модели.

2.2.4. Cheerp

Cheerp [3] - альтернатива Emscripten, также использующая LLVM представление. Принципиальное отличие в контексте данной работы заключается в том, что данная технология не пытается эмулировать адресное пространство с помощью типизированного массива, а прово-

Критерии	**Mandreel**	**Emscripten**	**Cheerp (ex. Duetto)**	**CrossBridge**
Лицензия	Проприетарная	Свободная	Смешанная	Свободная
Полная поддержка синтаксиса C/C++	Да	Да	Нет	Да
Доступ к DOM, HTML5 из C/C++	Нет	Нет	Да	Нет
Модель памяти	Типизированный массив	Типизированный массив	JS-структуры	?
Развитое сообщество	Нет	Да	Да	Нет
Доступ к файлам через fopen()	?	Да	Нет	?
Целевая технология	JavaScript	JavaScript	JavaScript	Flash

Рис. 1: Сравнение систем автоматической трансляции

дит прямое преобразование типов из C/C++ в JavaScript, что в итоге положительно сказывается на быстродействие в веб-приложении.

2.2.5. Вывод

Сравнение приведено на рисунке № 1. Основной выбор производился между Cheerp и Emscripten. В результате выбор был сделан в сторону последнего, так как Emscripten полноценно поддерживает весь синтаксис C/C++ и обладает наиболее схожей с нативной моделью памяти, что в дальнейшем положительно сказалось на переносимости кода. При этом стоит отметить, что Emscripten - проект с достаточно большим сообществом и подробной документацией, что является важным критерием при разработке нестандартных решений.

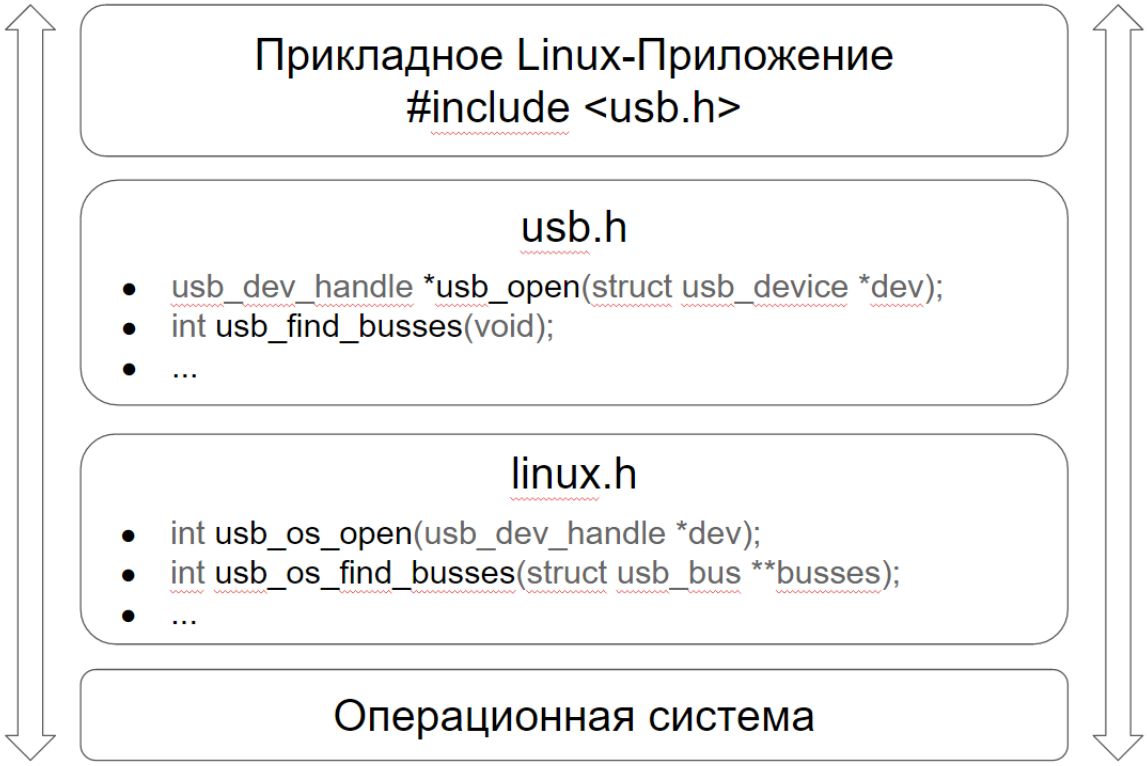

Рис. 2: Использование библиотеки libusb

2.3. Использование libusb прикладными Linux-приложениями

Libusb [23] - популярная linux-библиотека для прикладные приложения использующие в своей работе USB. В контексте данной работы под прикладными linux-приложениями понимаются специализированные приложения-загрузчики, которые используют libusb для подключения к отладочным платам. Использование библиотеки выглядит (рисунок № 2) следующим образом: все публичные структуры и функции, доступные прикладному приложению определены в заголовочном файле usb.h, а реализация в файле usb.c. Так как libusb может использоваться на разных платформах, то при сборке библиотеки происходит автоматическая подстановка платформо-зависимой реализации. В случае с linux'ом, это заголовочный файл linux.h и реализация linux.c. Поэтому для портирования библиотеки на новую платформу требуется разработать платформо-зависимые системные вызовы, которые в дальнейшем будут использоваться прикладным приложением.

2.4. Существующие решения

В настоящий момент существует несколько веб-приложения, использующие низкоуровневые интерфейсы для загрузки прошивок на отладочные платы. Наиболее известные из них - это ”ChromeDuiono”[12] и ”Codebender App”[2]. Приложения написаны “с нуля” на JavaScript, те разработчикам было необходимо реализовать механизм загрузки прошивки на отладочную плату самостоятельно. При этом требовалось учитывать, что на различных семействах контроллерах может быть разный алгоритм загрузки прошивки.

Преимущество моего решения в том, что я предлагаю не разрабатывать алгоритм прошивки контроллера каждый раз “с нуля”, а использовать автоматическую трансляцию уже готовых Linux-приложений. Разработчику приложения остается только реализовать пользовательский интерфейс.

3. Описание решения

3.1. Портирование libusb для работы в браузере

Под портированием понимается адаптация программного обеспечения для условий другой среды, нежели той для которой оно изначально разрабатывалось с сохранением функциональности [22]. В контексте данной работы это адаптация libusb к использованию Chrome USB API [9]. Как уже было сказано в разделе 2 библиотека разделена на два принципиальных блока: платформо-зависимого (который работает везде одинаково) и платформо-независимого (который меняется в зависимости от используемой платформы). Для портирования требуется изменить только платформо-зависимую часть, при этом внешний интерфейс должен остаться без изменений. Платформо-зависимая часть библиотеки libusb состоит из ряда функций:

```
static int device_open(struct usb_device *dev);
int usb_os_open(usb_dev_handle *dev);
int usb_os_close(usb_dev_handle *dev);
int usb_set_configuration(usb_dev_handle *dev, int configuration);
int usb_release_interface(usb_dev_handle *dev, int interface);
int usb_bulk_write(usb_dev_handle *dev, int ep, char *bytes,
    int size, int timeout);
int usb_interrupt_write(usb_dev_handle *dev, int ep, char *bytes,
    int size, int timeout);
...
```

Эти функции требуется изменить для работы с Chrome USB API. К примеру, функция открытия USB-устройства usb_os_open для Linux-версии библиотеки выглядит следующим образом:

```
static int device_open(struct usb_device *dev)
{
  char filename[PATH_MAX + 1];
  int fd;

  snprintf(filename, sizeof(filename) - 1, "%s/%s/%s",
    usb_path, dev->bus->dirname, dev->filename);
```

```
  fd = open(filename, O_RDWR);
  if (fd < 0) {
    fd = open(filename, O_RDONLY);
    if (fd < 0)
      USB_ERROR_STR(-errno, "failed to open %s: %s",
        filename, strerror(errno));
  }

  return fd;
}

int usb_os_open(usb_dev_handle *dev)
{
  dev->fd = device_open(dev->device);

  return 0;
}
```

Для работы с Chrome USB API библиотека была портирована. А именно вызов *usb_os_open( )* выглядит следующим образом:

```
int usb_os_open(usb_dev_handle *dev)
{
  #ifdef DEBUG
    EM_ASM({console.log("usb_os_open start")});
  #endif

  ready_to_continue = FALSE;
  usb_os_open_st();
  usb_cb_wait();
  dev->fd = g_handle;

  return 0;
}
```

При этом, внутренний принцип работы был достаточно сильно изменен из-за особенностей работы JavaScript в браузере. В данной функции важное значение имеют три строчки:
ready_to_continue = FALSE; - опускание флага, говорящий о том, что программа готова продолжить работу.

usb_os_open_st( ); - вызов функции, отвечающей за начало взаимодействия с Chrome USB API.
usb_cb_wait( ); - вызов функции, отвечающей за завершение взаимодействия с Chrome USB API.

Функция *usb_os_open_st( )* выглядит следующим образом:

```
void usb_os_open_st()
{
  int pid = mdev->device->descriptor.idProduct;
  int vid = mdev->device->descriptor.idVendor;
  int did = mdev->device->devnum;

  EM_ASM_(
  {
    var device = {};
    device["device"] = $0;
    device["manufacturerName"] = "";
    device["productId"] = $2;
    device["productName"] = "";
    device["serialNumber"] = "";
    device["vendorId"] = $1;

    var usb_os_open_cb = Module.cwrap
    ('usb_os_open_cb',
     'number',
     ['number']);
            chrome.usb.openDevice(device, function(handle)
              {
                if (chrome.runtime.lastError)
                {
                  console.warn(chrome.runtime.lastError.message);
                }
                  usb_os_open_cb(handle['handle']);
              });


  },did,vid,pid);
}
```

В данной функции используется команда *EM_ASM_*. Это аналог

вставки ассемблерного кода, иногда используемого при разработке на языке C. Только в данном случае происходит вставка кода на языке JavaScript. Внутри вставки сначала определяется служебная структура *device*, в которую записываются значения переменных, переданных из внешнего C окружения: идентификаторы производителя и устройства, а так же внутренний идентификатор, необходимый для Chrome USB API.

Затем следует вызов *Module.cwrap*, возвращающий функцию, которая является оберткой вокруг C-функции. Это один из способов доступа к C-функциям из JavaScript-кода. Далее происходит вызов функции из Chrome USB API *openDevice*. Первым аргументом которой является структура, заданная ранее. Вторым - функция обратного вызова, которая будет вызвана по завершению выполнения команды. В данном примере это анонимная функция, которая проверяет результат на ошибки и вызывает обертку над C-функцией (*usb_os_open_cb( )*), обрабатывающей полученные данные. Эта функция выглядит следующим образом:

```
void usb_os_open_cb(int handle_id)
{
  g_handle = handle_id;
  ready_to_continue = TRUE;
}
```

Она сохраняет полученное от Chrome USB API значение и поднимает флаг, отвечающий за готовность программы продолжить работу. Данный флаг используется в функции *usb_cb_wait( )*:

```
void usb_cb_wait(void)
{
  while(!ready_to_continue)
  {
    emscripten_sleep_with_yield(200);
  }
}
```

В этой функции реализован цикл, условием выхода из которого является поднятый флаг *ready_to_continue*. Внутри цикла используется

команда Emscripten. Она необходима для реализации синхронности работы библиотеки в условиях браузера. Поток выполнения JavaScript в браузере является полностью асинхронным и реализация "бесконечных" циклов с проверкой на какое-либо условие невозможна. Данная проблема решается с помощью *emscripten_sleep_with_yield( )*. Эта функция сохраняет текущий контекст и отдает поток управления браузеру на указанное количество времени (в моем примере, на 200 миллисекунд). При этом функции обратного вызова не запрещаются и продолжают работать. После истечения времени восстанавливается контекст и поток выполнения возвращается к коду. В данном случае только для проверки текущего статуса флага. Если во время работы основного потока браузера произошел вызов функции обратного вызова от Chrome USB API, то флаг будет поднят и после очередной проверки флага программа продолжит исполнение. Другими словами был реализован синхронный вызов функции *usb_os_open( )*.

С помощью описанного выше метода было произведено портирование всех необходимых функций платформо-зависимого уровня для работы в Chrome USB API. В результате, с помощью реализованного уровня, библиотека предоставляла необходимые интерфейсы прикладному приложению.

3.2. Внедрение libusb в Emscripten

В ходе исследования, описанного в разделе 2.2, была выбрана система автоматической source-to-source [15] трансляции Emscripten. Для того чтобы портировать приложение, написанное на C, требуется модифицировать рецепт сборки makefile [20]. Необходимо заменить стандартный компилятор на транслятор Emscripten и включить опциональные настройки, если это требуется. Для внедрения libusb необходимо отключить ключ компилятора -lusb, отвечающий за использование стандартной версии libusb, заменив его ключами, подключающими портированную версию libusb. Теперь в ходе трансляции Emscripten подставит нужную версию библиотеки. В результате приложение трансли-

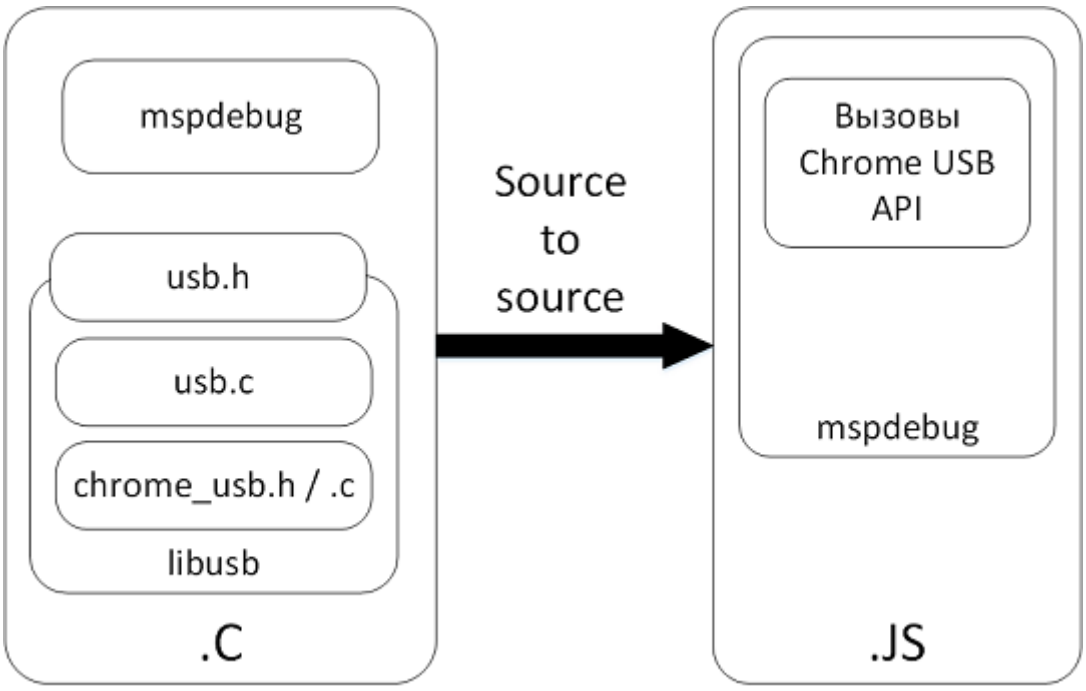

Рис. 3: Source-to-source трансляция

руется в JavaScript код и в тот момент, когда ему требуется обратится к libusb, оно будет обращается к Chrome USB API.

3.3. Демонстрационное приложение на базе mspdebug

В качестве апробации работы портированной версии libusb, была произведена автоматическая трансляция прикладного Linux-приложения mspdebug [24]. Это специальное приложения для работы с отладочными платами семейства MSP430 [14], производства Texas Instruments [13]. Mspdebug упаковано в веб-приложение и позволяет загружать прошивки на микроконтроллеры из браузера. Работоспособность приложения проверена в браузере Chrome на операционных системах Ubuntu, Mac OS и Windows. На рисунке № 4 показан пример работы портированной версии mspdebug, где стандартный вывод stdout переключен на Javascript-консоль в браузере.

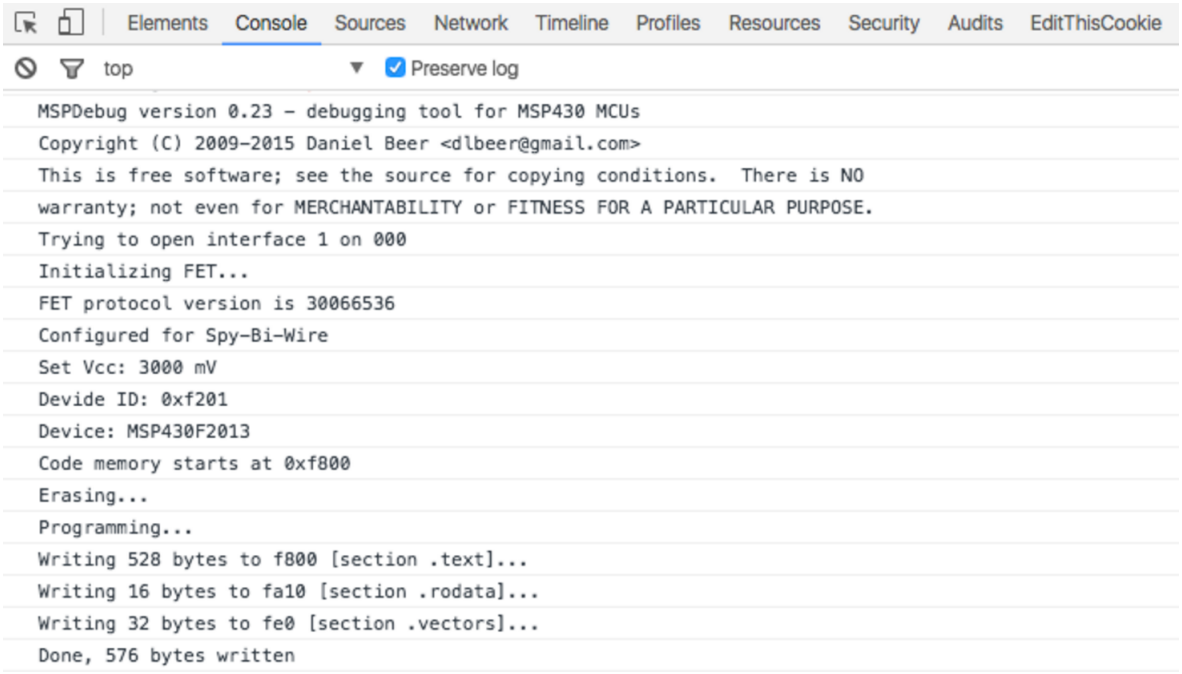

Рис. 4: Работа mspdebug в браузере

Заключение

В рамках данной работы была реализована возможность автоматической трансляции прикладных Linux-приложений, использующих только системные вызовы USB, в веб-приложения. Были выполнены следующие задачи:

- портирована libusb для работы в браузере через Chrome USB API;
- внедрена портированная версия libusb в систему автоматической трансляции Emscripten;
- разработано демонстрационное веб-приложение на базе mspdebug.

Список литературы

[1] Adobe. CrossBridge. — 2016. — URL: http://adobe-flash.github.io/crossbridge/ (дата обращения: 24.05.2016).

[2] CC Codebender. Codebender App page. — 2016. — URL: https://chrome.google.com/webstore/detail/codebender-app/magknjdfniglanojbpadmpjlglepnlko (дата обращения: 21.05.2016).

[3] Cheerp. Official site. — 2016. — URL: http://leaningtech.com/cheerp/ (дата обращения: 24.05.2016).

[4] Cisco. Cisco IoT press realese. — 2014. — URL: http://www.cisco.com/web/RU/news/releases/txt/2014/10/102014e.html (дата обращения: 11.04.2016).

[5] Emscripten. Official site. — 2016. — URL: http://kripken.github.io/emscripten-site/ (дата обращения: 24.05.2016).

[6] Foundation Mozilla. WebAPI/WebUSB. — 2016. — URL: https://wiki.mozilla.org/WebAPI/WebUSB (дата обращения: 16.05.2016).

[7] Gartner. Gartner, IoT reasearch. — 2016. — URL: http://www.gartner.com/technology/research/internet-of-things/ (дата обращения: 07.04.2016).

[8] Google. Chrome APP API. — 2016. — URL: https://developer.chrome.com/apps/api_index (дата обращения: 16.05.2016).

[9] Google. Chrome USB API. — 2016. — URL: https://developer.chrome.com/apps/usb (дата обращения: 11.04.2016).

[10] Google. Native client. — 2016. — URL: https://developer.chrome.com/native-client (дата обращения: 16.05.2016).

[11] Google. Native messaging. — 2016. — URL: https://developer.chrome.com/extensions/nativeMessaging (дата обращения: 16.05.2016).

[12] Halverson Casey. ChromeDuino page. — 2016. — URL: https://chrome.google.com/webstore/detail/chromeduino/dmkincdpchiadkhhocmbpjljebfifgbl (дата обращения: 21.05.2016).

[13] Instrument Texas. Official site. — 2016. — URL: http://www.ti.com/ (дата обращения: 11.04.2016).

[14] MSP430. Official page. — 2016. — URL: http://www.ti.com/ww/en/launchpad/launchpads-msp.html (дата обращения: 11.04.2016).

[15] Plaisted David A. Source-to-Source Translation and Software Engineering // Journal of Software Engineering and Applications, 2013, 6, 30-40. — 2013.

[16] W3C. HTML5 specification. — 2016. — URL: https://www.w3.org/TR/html5 (дата обращения: 16.05.2016).

[17] W3C. WebUSB API Specification. — 2016. — URL: https://wicg.github.io/webusb/ (дата обращения: 16.05.2016).

[18] Wikipedia. ActiveX. — 2016. — URL: https://en.wikipedia.org/wiki/ActiveX (дата обращения: 16.05.2016).

[19] Wikipedia. Chrome OS. — 2016. — URL: https://en.wikipedia.org/wiki/Chrome_OS (дата обращения: 11.04.2016).

[20] Wikipedia. Make (build automation tool). — 2016. — URL: https://en.wikipedia.org/wiki/Make_(software) (дата обращения: 11.04.2016).

[21] Wikipedia. NPAPI. — 2016. — URL: https://en.wikipedia.org/wiki/NPAPI (дата обращения: 16.05.2016).

[22] Wikipedia. Software porting. — 2016. — URL: https://en.wikipedia.org/wiki/Porting (дата обращения: 11.04.2016).

[23] libusb. Official site. — 2016. — URL: http://www.libusb.org/ (дата обращения: 11.04.2016).

[24] mspdebug. Official site. — 2016. — URL: http://dlbeer.co.nz/mspdebug/ (дата обращения: 11.04.2016).